REMARKS

Claims 1-37 are pending.

Rejections of Claims 1 – 37

Claims 1 – 37 stand rejected under 35 U.S.C. §102(b) as being unpatentable over "Java Security," by *Oaks*. Applicant traverses these rejections for at least the following reasons, and respectfully requests that the rejections be reconsidered and withdrawn.

Before addressing each of the rejections in detail, the present Application and *Oaks* are briefly discussed. Fundamentally, Oaks does not disclose permission requests, as permission requests are described in the present Application. As described in the present Application, a permission request defines characteristics of permissions related to a code assembly. The permission request may specify a permission that is required by, optional to, or not to be granted to, a code assembly. As such, the permission requests can be used to filter the permission policy that will be applied to a demand from the code assembly to access a resource.

By contrast, *Oaks* describes a permission object that enables a code assembly to actually ask for permission to access a resource. As *Oaks* describes, "a permission object represents an actual permission that has been granted to that class," or "a permission object allows us to ask if we have a specific permission." *Oaks* provides a user-generated policy against which the permission object is

checked to determine if a code assembly has a specific permission. As such, *Oaks* does not describe permission requests that can be used to filter the policy that is ultimately used to evaluate an actual demand for permission.

With the foregoing in mind, each of the claims is now discussed in detail. Claim 1 is reproduced here:

> 1. A method for processing a permission set associated with a code assembly received from a resource location to control execution of the code assembly, the method comprising:
>
> receiving the permission set including at least one permission associated with the code assembly;
>
> receiving a permission request set in association with the code assembly; and
>
> filtering the permission set based on the permission request set to control execution of the code assembly.

Claim 1 recites, in part, filtering a permission set based on a permission request set to control execution of a code assembly. The Office relies on *Oaks*, pages 90 – 93 and 100, in its rejection of claim 1. In general, the *Oaks* system enables administrators and end users to define in a file (i.e., the PolicyFile), the security policy that is applied to each code assembly. See *Oaks* at p. 110. The policy file maps code sources to sets of permissions. See *id* at 112. When a

permission is requested, the requested permission is checked against the policy file to determine if the associated operation should be permitted. See *id* at 116.

Oaks' system is akin to conventional approaches discussed in the Background of the present Application. For example, *Oaks'* policy file includes entries such as:

```
grant  codeBase  http://www.xyz.com/  {
      permission java.io.FilePermssion "${user.home}${/}docs${/}-
",  "read, write, delete";};
```

The foregoing entry means that any code loaded from the top-level directory of *www.xyz.com* is granted permission to use any files under the user's *docs* directory. See *Oaks* at p. 112. *Oaks'* approach corresponds closely to the example described in the *Background* of the present *Application* at page 2, lines 11 – 24. As discussed in the *Background*, in such conventional approaches, security policies are static, remaining fixed over long periods of time. See *Application*, p. 3.

By contrast, claim 1 includes filtering a permission set based on a permission request set. *Oaks'* policy file is not filtered based on a permission request set. Applicants' have thoroughly reviewed *Oaks'* disclosure and can find no teaching of filtering a permission set (or a security policy) based on a permission request set. For at least this reason, *Oaks* fails to teach or suggest all of the elements of claim 1.

Claims 2 – 19 each depend from claim 1 in some manner. Therefore claims

2 – 19 are believed to be allowable for at least the same reasons as claim 1. In addition, each of claims 2 – 19 has its own limitations that further distinguish it from *Oaks*.

For example, claim 2 is reproduced here:

> 2. The method of claim 1 wherein the filtering operation comprises:
>
> generating a permission grant set from a subset of the permission set, the subset specified by the permission request set.

Claim 2 recites, in part, generating a permission grant set from a subset of the permission set, the subset specified by the permission request set. *Oaks'* security policy (i.e., the PolicyFile) includes grant entries that map permissions to particular code sources. The grant entries are constructed by hand by an administrator or an end user. See *Oaks* at p. 110 and 114. Because the grant entries are constructed by hand, the grant entries cannot be filtered based on a permission request set associated with a code assembly. Thus, *Oaks'* grant entries are not generated from a subset of a permission set that is specified by a permission request set associated with a code assembly.

The Office relies on page 101 of *Oaks* in its rejection of claim 2. Page 101 of *Oaks* describes an example of how a permission can be created, but not how a

grant entry is created in the policy file. *Oaks'* "Permission class is really used only to create your own types of permissions." See *Oaks* at 99 (emphasis added). The exemplary code on page 101 shows how a Permission class is created; that is, by creating the name of the Permission class (e.g., XYZPayrollPermission), and the types of actions defined for the permission (e.g., "view", and "update"). *Oaks'* permission object is used to ask if a code assembly has a specific permission. See *id* at 93. The permission name and action are used at run-time to check against the policy file to determine if the particular permission can perform the requested action. This is done by calling the "`checkPermission()`" method. See *id* at 117.

For at least the foregoing reasons claim 2 is neither disclosed nor suggested by *Oaks*.

In addition, claim 19 recites:

> 19. The method of claim 1 wherein the operation of receiving a permission request set comprises:
>
> retrieving the permission request set in a network communication distinct from a network communication in which the code assembly is received.

The Office asserts that retrieving a permission request set in a network communication distinct from a network communication in which the code

assembly is received is disclosed at page 93 in *Oaks*. Applicants have thoroughly reviewed page 93 of *Oaks* but are unable to find a disclosure of the subject matter of claim 19. Under the heading "The CodeSource Class", *Oaks* discusses that there could be a different code source for each class. However, *Oaks* does not discuss retrieving a permission request set in a network communication distinct from a network communication in which the code assembly is received, as is discussed in claim 19.

For at least the foregoing reasons claim 19 is neither disclosed nor suggested by *Oaks*.

Turning to claim 20, claim 20 is reproduced here:

> 20. A policy manager module for processing a permission set associated with a code assembly received from a resource location to control execution of the code assembly, the policy manager module comprising:
>
> a filter receiving the permission set and a permission request set associated with the code assembly and filtering the permission set based on the permission request set to control execution of the code assembly.

Claim 20 recites in part, a filter receiving a permission set and a permission request set associated with the code assembly and filtering the permission set based on the permission request set to control execution of the code assembly. As discussed above with regard to claim 1, *Oaks'* security policy is embodied in a policy file that is generated by a user or administrator. *Oaks'* policy file includes the grant entries that specify which code sources are granted what permissions. Although a user can direct which policy file is used, the grant entries of the policy file are not filtered. As discussed above, *Oaks'* static grant entries are checked against a permission request at run-time to determine whether a requested action can be taken.

For at least the foregoing reason, Oaks fails to teach or suggest all the limitations of claim 20. Claim 20 is therefore believed to be allowable.

Claims 21 – 32 each depends from claim 20 in some manner. Claims 21 – 32 are therefore believed to be allowable for at least the same reasons as claim 20.

Referring to claims 33 and 34, these claims are reproduced here:

> 33. A computer data signal embodied in a carrier wave by a computing system and encoding a computer program for executing a computer process processing a permission set associated with a code assembly received from a resource location to control execution of the code assembly, the computer process comprising:

receiving the permission set including at least one permission associated with the code assembly;

receiving a permission request set in association with the code assembly; and

filtering the permission set based on the permission request set to control execution of the code assembly.

34. A computer program storage medium readable by a computer system and encoding a computer program for executing a computer process processing a permission set associated with a code assembly received from a resource location, the computer process comprising:

receiving the permission set including at least one permission associated with the code assembly;

receiving a permission request set in association with the code assembly; and

filtering the permission set based on the permission request set to control execution of the code assembly.

Claims 33 and 34 both recite in part, filtering a permission set based on the permission request set to control execution of a code assembly associated with the

permission request set. As discussed above with regard to claim 1, *Oaks'* security policy is embodied in a policy file that is generated by a user or administrator. *Oaks'* policy file includes the grant entries that specify which code sources are granted what permissions. Although a user can direct which policy file is used, the grant entries of the policy file are not filtered. As discussed above, *Oaks'* static grant entries are checked against a permission request at run-time to determine whether a requested action can be taken.

For at least these reasons, Oaks fails to teach or suggest all the claim limitations of either of claims 33 and 34. Therefore, claims 33 and 34 are believed to be allowable.

Referring to claim 35, this claim is reproduced here:

> 35. A computer program product encoding a computer program for executing on a computer system a computer process processing a permission set associated with a code assembly received from a resource location to control execution of the code assembly, the computer process comprising:
>
> defining a code group collection based on a security policy specification, the code group collection including one or more code groups;
>
> receiving evidence associated with the code assembly;
>
> evaluating membership of the code assembly in the one or more code groups, based on the evidence;

generating the permission set based on the membership of the code assembly in the one or more code groups;

receiving the permission set including at least one permission associated with the code assembly;

receiving a permission request set in association with the code assembly; and

computing a logical set operation on the permission set and the permission request set to generate a permission grant set.

Claim 35 recites in part, computing a logical set operation on the permission set and the permission request set to generate a permission grant set. As discussed above, *Oaks'* permission grant entries are listed in a policy file that is generated by a user or system administrator, who map permissions to code sources. See *Oaks* at 112. The mapping of permissions to code sources involves simply specifying the appropriate permissions for a code source. Once these grant entries are defined in *Oaks'* policy file, they are static. Oaks neither discloses nor suggests generating the permission grant set by computing a logical set operation on the permission set and the permission request set.

For at least the foregoing reasons, Oaks fails to disclose or suggest all the elements of claim 35. As such, claim 35 is believed to be allowable.

Claims 36 – 37 each depend in some manner from claim 35. Therefore, claims 36 – 37 are believed to be allowable for at least the same reasons as claim 35.

Conclusion

Claims 1 - 37 are in condition for allowance and are patentable over the cited art. Accordingly, Applicant respectfully requests that a Notice of Allowability be issued forthwith. If the Office's next anticipated action is to be anything other than issuance of a Notice of Allowability, Applicant respectfully requests a telephone call for the purpose of scheduling an interview.

Respectfully Submitted,

Date: 9/17/04

By: _____
Damon A. Rieth
Reg. No. 52,167
(303) 539-0265 x 237